Machine Learning for NLP

Unsupervised Learning

Aurélie Herbelot

2021

Centre for Mind/Brain Sciences
University of Trento

Unsupervised learning

- In unsupervised learning, we learn without training data.
- The idea is to find a structure in the unlabeled data.
- The following unsupervised learning techniques are fundamental to NLP:
  - dimensionality reduction (e.g. PCA, using SVD or any other technique);
  - clustering;
  - *some* neural network architectures.

Dimensionality reduction

Dimensionality reduction

- Dimensionality reduction refers to a set of techniques used to reduce the number of variables in a model.
- For instance, we have seen that a count-based semantic space can be reduced from thousands of dimensions to a few hundreds:
  - We build a space from word co-occurrence, e.g. *cat - meow: 56* (we have seen *cat* next to *meow* 56 times in our corpus).
  - A complete semantic space for a given corpus would be a $N \times N$ matrix, where N is the size of the vocabulary.
  - N could be well in the hundreds of thousands of dimensions.
  - We typically reduce N to 300-400.

From PCA to SVD

- We have seen that Principal Component Analysis (PCA) is used in the Partial Least Square Regression algorithm for *supervised learning*.
- PCA is *unsupervised* in that it finds ‘the most important’ dimensions in the data just by finding structure in that data.
- A possible way to find the principal components in PCA is to perform Singular Value Decomposition (SVD).
- Understanding SVD gives an insight into the nature of the principal components.

Singular Value Decomposition

- SVD is a **matrix factorisation** method which expresses a matrix in terms of three other matrices:

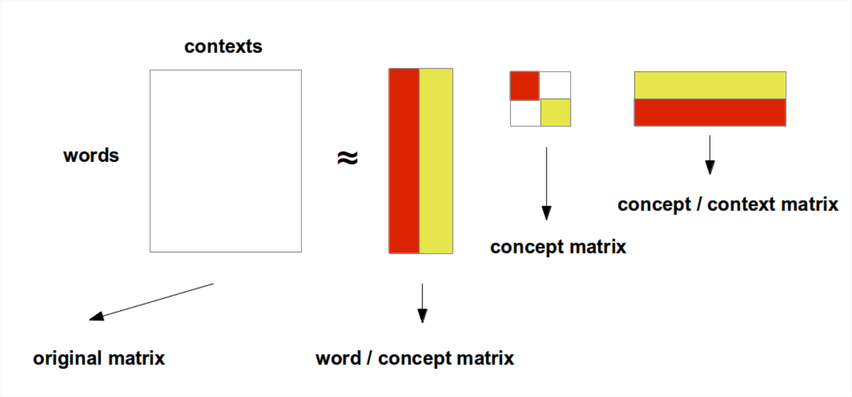
$$A = U\Sigma V^T$$

- U and V are orthogonal: they are matrices such that
  - $UU^T = U^TU = I$
  - $VV^T = V^TV = I$

  I is the identity matrix: a matrix with 1s on the diagonal, 0s everywhere else.

- Σ is a diagonal matrix (only the diagonal entries are non-zero).

Singular Value Decomposition over a semantic space



Taking a linguistic example from distributional semantics, the original word/context matrix A is converted into three matrices U, Σ, V^T, where contexts have been aggregated into ‘concepts’.

The SVD derivation

- From our definition, $A = U\Sigma V^T$, it follows that...
- $A^T = V\Sigma^T U^T$

  See https://en.wikipedia.org/wiki/Transpose for explanation of transposition.
- $A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$
  Recall that $U^T U = I$ because U is orthogonal.
- $A^T A V = V\Sigma^2 V^T V = V\Sigma^2$
  Since $V^T V = I$.
- Note the V on both sides: $A^T A V = V\Sigma^2$
- (By the way, we could similarly prove that $AA^T U = U\Sigma^2$...)

SVD and eigenvectors

- Eigenvectors again! The *eigenvector* of a linear transformation doesn't change its direction when that linear transformation is applied to it:

$$Av = \lambda v$$

A is the linear transformation, and λ is just a scaling factor: v becomes 'bigger' or 'smaller' but doesn't change direction. v is the eigenvector, λ is the eigenvalue.

- Let's consider again the end of our derivation: $A^T AV = V\Sigma^2$.
- This looks very much like a linear transformation applied to its eigenvector (but with matrices)...

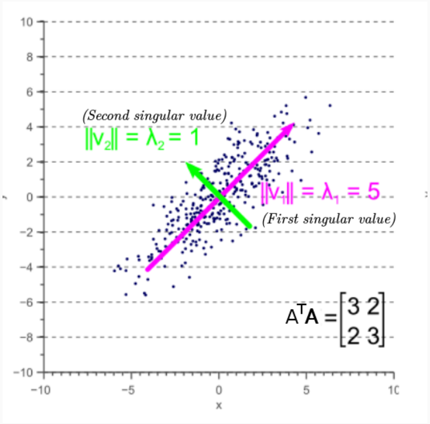
NB: $A^T A$ is a square matrix. This is important, as we would otherwise not be able to obtain our eigenvectors.

SVD and eigenvectors

- The columns of V are the eigenvectors of $A^T A$.
  (Similarly, the columns of U are the eigenvectors of AA^T.)
- $A^T A$ computed over *normalised data* is the *covariance* matrix of A.
  See https://datascienceplus.com/understanding-the-covariance-matrix/.
- In other words, each column in V / U captures variance along one of the (possibly rotated) dimensions of the n-dimensional original data (see last week's slides).

The singular values of SVD

- Σ itself contains the eigenvalues, also known as *singular values*.
- The top k values in Σ correspond to the spread of the variance in the top k dimensions of the (possibly rotated) eigenspace.



http://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/

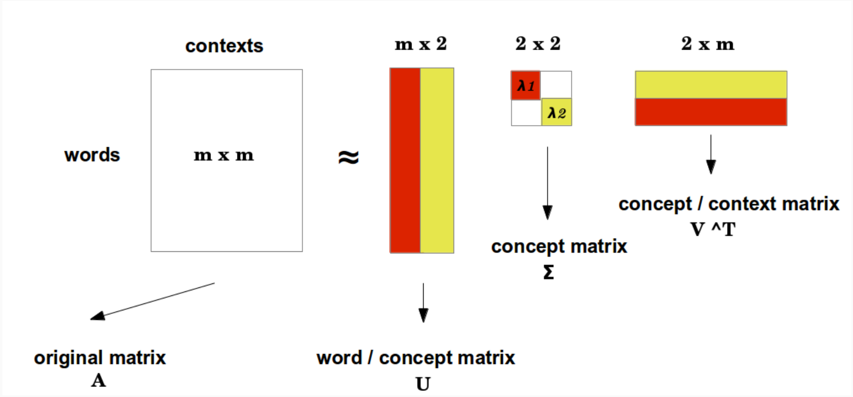
SVD at a glance

- Calculate A^TA = covariance of input matrix A (e.g. word / context matrix).
- Calculate the eigenvalues of A^TA. Take their square roots to obtain the singular values of A^TA (i.e. the matrix Σ).
  If you want to know how to compute eigenvalues, see http://www.visiondummy.com/2014/03/eigenvalues-eigenvectors/.
- Use the eigenvalues to compute the eigenvectors of A^TA. These eigenvectors are the columns of V.
- We had set $A = U\Sigma V^T$. We can re-arrange this equation to obtain $U = AV\Sigma^{-1}$.

Finally... dimensionality reduce!

- Now we know the value of U, Σ and V.
- To obtain a reduced representation of A, choose the top k singular values in Σ and multiply the corresponding columns in U by those values.
- We now have A in a k-dimensional space corresponding to the dimensions of highest covariance in the original data.

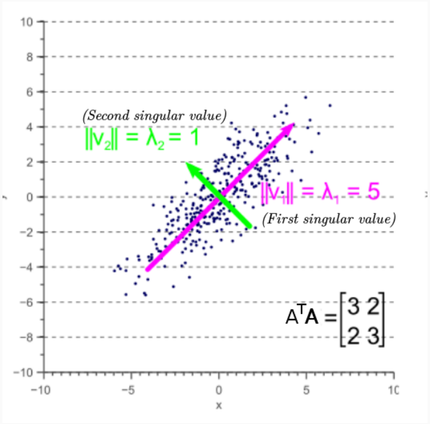
Singular Value Decomposition



What semantic space?

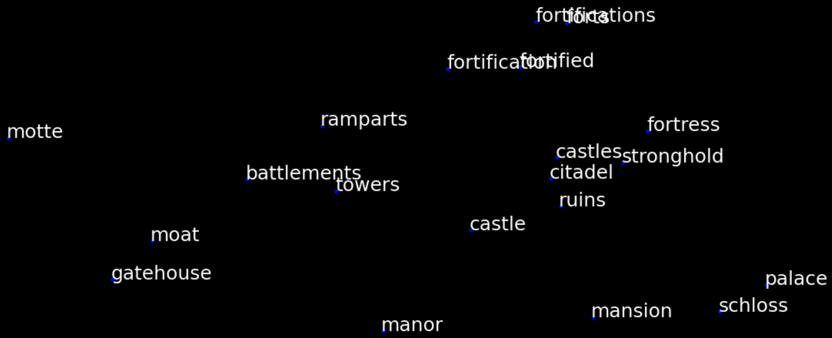
- Singular Value Decomposition (LSA – Landauer and Dumais, 1997). A new dimension might correspond to a generalisation over several of the original dimensions (e.g. the dimensions for *car* and *vehicle* are collapsed into one).
    - + Very efficient (200-500 dimensions). Captures generalisations in the data.
    - - SVD matrices are not straightforwardly interpretable. Can you see why?

The SVD dimensions



Say that in the original data, the x-axis was the context *cat* and the y-axis the context *chase*, what is the purple eigenvector?

PCA for visualisation



Random indexing

Distributional Semantics and Random indexing

Count-based models

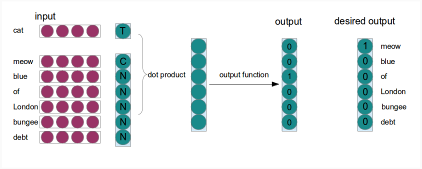
... the walls of the *castle* were built...
... the siege of the *castle* lasted...
... count moved to the *castle* in...

	wall	siege	whale	...
castle	1	1	0	...

Random indexing (RI) is a technique to build distributional semantics space at reduced dimensionality. It works on **count-based** models.

✓

Predictive models



Compare with ‘predictive’ techniques using NNs, which also build a space at reduced dimensionality by virtue of the dimensionality of their input layer.

Why random indexing?

- No distributional semantics method so far satisfies all ideal requirements of a *semantics acquisition model*:
  1. show human-like behaviour on linguistic tasks;
  2. have low dimensionality for efficient storage and manipulation ;
  3. be efficiently acquirable from large data;
  4. be transparent, so that linguistic and computational hypotheses and experimental results can be systematically analysed and explained ;
  5. be **incremental** (i.e. allow the addition of new context elements or target entities). Both standard count models and predictive models fail in that respect.

Random Indexing: the basic idea

- We want to derive a semantic space S by applying a random projection R to a matrix of co-occurrence counts M:

$$M_{p \times n} \times R_{n \times k} = S_{p \times k}$$

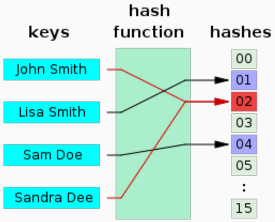
- We assume that $k << n$. So this has in effect dimensionality-reduced the space.
- The above projection can be implemented as an *incremental* process, allowing us to build semantic spaces ‘in real time’.

The components of Random Indexing

- **Random Indexing** uses the principle of **Locality Sensitive Hashing.**
- So we will look in turn at:
  - conventional hashing, and why it does not cater for similarity;
  - the definition of LSH, which solves the similarity problem;
  - how LSH implements the idea of a projection matrix;
  - how a projection can be decomposed into an *incremental* addition operation.

Hashing: definition

- Hashing is the process of converting data of arbitrary size into fixed size signatures (number of bytes).
- The conversion happens through a *hash function*.
- A *collision* happens when two inputs map onto the same *hash (value).*
- Since multiple values can map to a single hash, the slots in the hash table are referred to as *buckets*.



https://en.wikipedia.org/wiki/Hash_function

Hashing strings: an example

- An example function to hash a string s:

$$s[0] * 31^{n-1} + s[1] * 31^{n-2} + ... + s[n-1]$$

  where $s[i]$ is the ASCII code of the ith character of the string and n is the length of s.
- This will return an integer.

Hashing strings: an example

- An example function to hash a string s:

$$s[0] * 31^{n-1} + s[1] * 31^{n-2} + ... + s[n-1]$$

- **A test:** 65 32 84 101 115 116 Hash: 1893050673
- **a test:** 97 32 84 101 115 116 Hash: 2809183505
- **A tess:** 65 32 84 101 115 115 Hash: 1893050672

Modular hashing

- Modular hashing is a very simple hashing function with high risk of collision:

$$h(k) = k \; mod \; m$$

- Let's assume a number of buckets $m = 100$:
  - h(A test) = h(1893050673) = 73
  - h(a test) = h(2809183505) = 5
  - h (a tess) = h(1893050672) = 72

- NB: no notion of similarity between inputs and their hashes. *A test* and *a tess* are very similar but *a test* and *a tess* are not.

Locality Sensitive Hashing

- In ‘conventional’ hashing, similarities between datapoints are not conserved.
- LSH is a way to produces hashes that can be compared with a similarity function.
- The hash function is a projection matrix defining a *random* hyperplane. If the projected datapoint $\vec{v}$ falls on one side of the hyperplane, its hash $h(\vec{v}) = +1$, otherwise $h(\vec{v}) = -1$.

Locality Sensitive Hashing

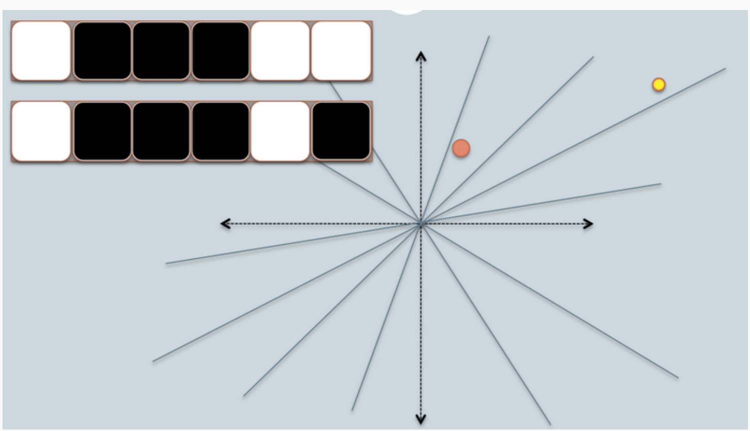
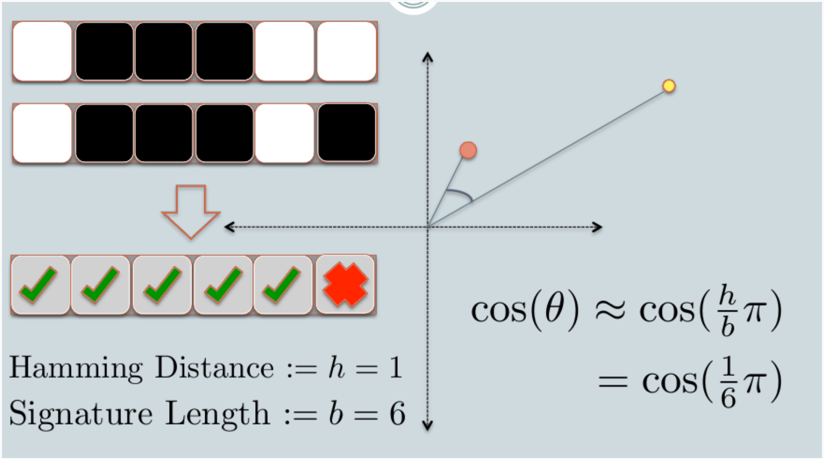


Image from VanDurme & Lall (2010): http://www.cs.jhu.edu/ vandurme/papers/VanDurmeLallACL10-slides.pdf

Locality Sensitive Hashing



$\cos(\theta) \approx \cos(\frac{h}{b}\pi)$
$= \cos(\frac{1}{6}\pi)$
Hamming Distance $:= h = 1$
Signature Length $:= b = 6$

Image from VanDurme & Lall (2010): http://www.cs.jhu.edu/ vandurme/papers/VanDurmeLallACL10-slides.pdf
(The *Hamming distance* between two strings of equal length is the number of positions at which the symbols differ across strings.)

Doing LSH over a semantic space

- The hash value of an input point in LSH is made of all the projections on all chosen hyperplanes:

$$M_{p \times n} \times R_{n \times k} = S_{p \times k}$$

- Say we have k=10 hyperplanes $h_1 ... h_{10}$ (the columns of matrix R) and we are projecting the vector $\overrightarrow{dog}$ with dimensionality n=300 on those hyperplanes:
  - dimension 1 of the new vector is the dot product of $\overrightarrow{dog}$ and h_1: $\sum dog_i h_{1i}$
  - dimension 2 of the new vector is the dot product of $\overrightarrow{dog}$ and h_2: $\sum dog_i h_{2i}$
  - ...
- We end up with a ten-dimensional vector which is the hash of $\overrightarrow{dog}$.

Interpretation of the LSH hash

- Each hyperplane is a discriminatory feature cutting through the data.
- Each point in space is expressed as a function of those hyperplanes.
- We can think of them as new ‘dimensions’ relevant to explaining the structure of the data.
- Random indexing is ‘random’ because the matrix $R_{n \times k}$ is obtained by sampling vectors (hyperplanes) from a Gaussian distribution, in a way that each one is orthogonal to the others.

Random Indexing: incremental LSH

- A random indexing space can be simply and incrementally produced through a two-step process:
  1. Map each *context* item c in the text to a random projection vector (to get R).
  2. Initialise each *target* item t as a null vector. Whenever we encounter c in the vicinity of t we update $\vec{t} = \vec{t} + \vec{c}$.
- The method is extremely efficient, potentially has low dimensionality (we can choose the dimension of the projection vectors), and is fully incremental.

Relating vector addition to random projection

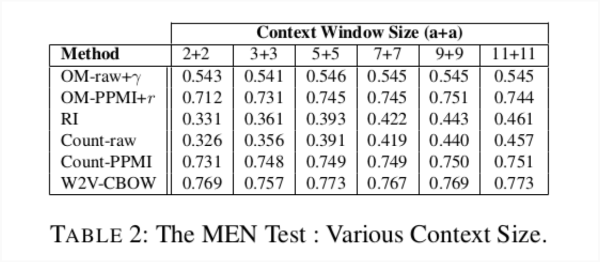
- We said that LSH used a random projection matrix so that $M_{p\times n} \times R_{n\times k} = S_{p\times k}$
- Here's a toy example.

$$\begin{array}{c} \\ t_1 \\ t_2 \end{array} \begin{array}{cc} c_1 & c_2 \\ \multicolumn{2}{c}{\begin{pmatrix} 2 & 3 \\ 1 & 0 \end{pmatrix}} \end{array} \times \begin{array}{c} r \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{array} \begin{array}{c} \\ c_1 \\ c_2 \end{array} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

- So we have two random vectors (0) and (1) corresponding to contexts c_1 and c_2.
- We got 3 by computing $2 \times 0 + 3 \times 1$
- This is equivalent to computing $(0 + 0) + (1 + 1 + 1)$
- **So we added the random vectors corresponding to each context for each time it occurred with the target. That's incremental random indexing.**

Is RI human-like?

- Not without adding PPMI weighting at the end of the RI process... (This kills incrementality.)



	Context Window Size (a+a)					
Method	2+2	3+3	5+5	7+7	9+9	11+11
OM-raw+γ	0.543	0.541	0.546	0.545	0.545	0.545
OM-PPMI+r	0.712	0.731	0.745	0.745	0.751	0.744
RI	0.331	0.361	0.393	0.422	0.443	0.461
Count-raw	0.326	0.356	0.391	0.419	0.440	0.457
Count-PPMI	0.731	0.748	0.749	0.749	0.750	0.751
W2V-CBOW	0.769	0.757	0.773	0.767	0.769	0.773

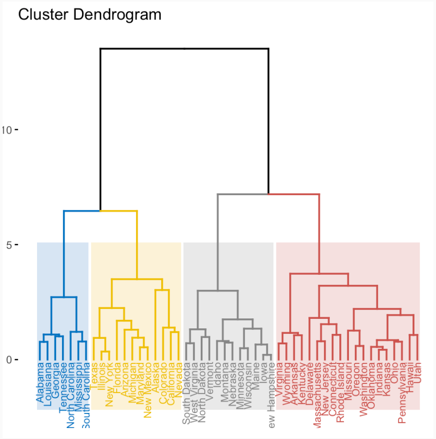
TABLE 2: The MEN Test : Various Context Size.

QasemiZadeh et al (2017)

Clustering

Clustering algorithms

- A clustering algorithm partitions some objects into groups named *clusters.*
- Objects that are similar according to a certain set of features should be in the same cluster.



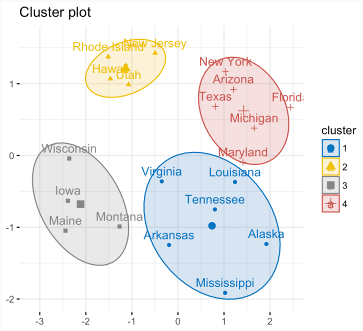
From http://www.sthda.com/english/articles/25-cluster-analysis-in-r-practical-guide/

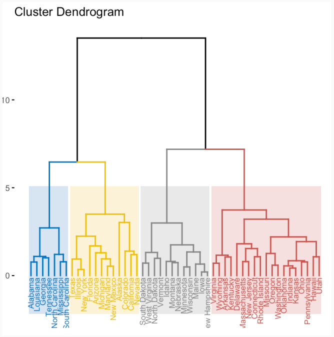
Why clustering

- **Example:**[1] we are translating from French to English, and we know from some training data that:
  - *Dimanche* $\longrightarrow$ *on Sunday*;
  - *Mercredi* $\longrightarrow$ *on Wednesday*;
- What might be the correct preposition to translate *Vendredi* into the English ___ *Friday*? (Given that we haven't seen it in the training data.)
- We can assume that the days of the week form a semantic cluster, which behave in the same way syntactically. Here, clustering helps us **generalise**.

[1] Example from Manning & Schütze, *Foundations of statistical language processing.*

Flat vs hierarchical clustering





Flat clustering

Hierarchical clustering

From http://www.sthda.com/english/articles/25-cluster-analysis-in-r-practical-guide/

Soft vs hard clustering

- In hard clustering, each object is assigned to only one cluster.
- In soft clustering, assignment can be to multiple clusters, or be probabilistic.
- In a *probabilistic* setup, each object has a probability distribution over clusters. $P(c_k|x_i)$ is the probability that object x_i belongs to cluster c_k.
- In a *vector space*, the degree of membership of an object x_i to each cluster can be defined by the similarity of x_i to some representative point in the cluster.

Centroids and medoids

- The *centroid* or *center of gravity* of a cluster c is the average of its N members:

$$\mu_k = \frac{1}{N} \sum_{x \in c_k} x$$

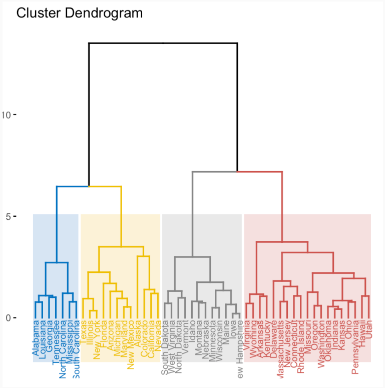
- The *medoid* of a cluster c is a prototypical member of that cluster (its average distance to all other objects in c is minimal):

$$x_{\text{medoid}} = \text{argmin}_{y \in \{x_1, x_2, \cdots, x_n\}} \sum_{i=1}^{n} d(y, x_i)$$

Hierarchical clustering: bottom-up

- Bottom-up agglomerative clustering is a form of hierarchical clustering.
- Let's have n datapoints $x_1 ... x_n$. The algorithm functions as follows:
    - Start with one cluster per datapoint: $c_i := \{x_i\}$.
    - Determine which two clusters are the most similar, and merge them.
    - Repeat until we are left with only one cluster $C = \{x_1 ... x_n\}$.

Hierarchical clustering: bottom-up



NB: The order in which clusters are merged depends on the similarity distribution amongst datapoints.

Hierarchical clustering: top-down

- Top-down divisive clustering is the pendent of agglomerative clustering.
- Let's have *n* datapoints again: $x_1 ... x_n$. The algorithm relies on a *coherence* and a *split* function:
  - Start with a single cluster $C = \{x_1 ... x_n\}$
  - Determine the least coherent cluster and split it into two new clusters.
  - Repeat until we have one cluster per datapoint: $c_i := \{x_i\}$.

Coherence

- **Coherence** is a measure of the pairwise similarity of a set of objects.
- A typical coherence function:

$$Coh(x_{1...n}) = mean\{Sim(x_i, x_j), ij \in 1...n, i < j\}$$

- E.g. coherence may be used to calculate the consistency of topics in topic modelling.

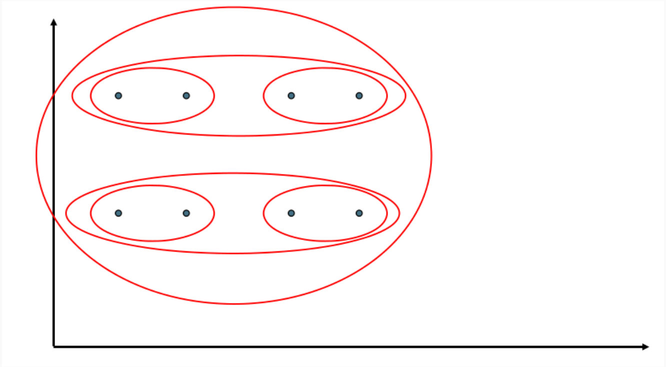
Split

- We need a heuristic to split a cluster into two new clusters.
- Take the point with the *maximum* distance to all other points in the cluster c_n and make it a new cluster c_{n+1}.
- Move all points in c_n which are more similar to c_{n+1}.

Similarity functions for clustering

- **Single link:** similarity of two most similar objects across clusters.
- **Complete link:** similarity of two least similar objects across clusters.
- **Group-average:** average similarity between objects. (Here, the average is over all pairs, within and across clusters.)

Effect of similarity function

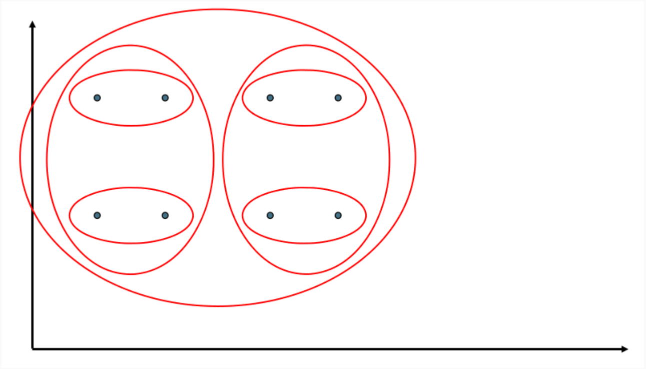


Clustering with single link

Graph taken from Manning, Raghavan & Schütze:

http://www.cs.ucy.ac.cy/courses/EPL660/lectures/lecture12-clustering.pdf

Effect of similarity function



Clustering with complete link

Graph taken from Manning, Raghavan & Schütze:
http://www.cs.ucy.ac.cy/courses/EPL660/lectures/lecture12-clustering.pdf

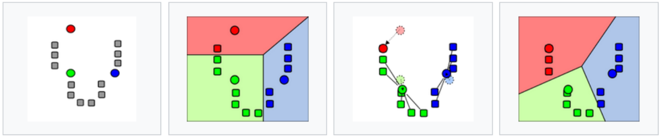
K-means clustering

- K-means is the main flat clustering algorithm.
- Goal in K-means: minimise the *residual sum of squares (RSS)* of objects to their cluster centroids:

$$RSS_k = \sum_{x \in c_k} |x - \mu_k|^2$$

- The intuition behind using RSS is that good clusters should have a) small intra-cluster distances; b) large inter-cluster distances.

K-means algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Source: https://en.wikipedia.org/wiki/K-means_clustering

Convergence

- Convergence happens when the RSS will not decrease anymore.
- It can be shown that K-means will converge to a local minimum, but not necessarily a global minimum.
- Results will vary depending on seed selection.

Initialisation

- Various heuristics to ensure good clustering:
  - exclude outliers from seed sets;
  - try multiple intialisations and retain the one with lowest RSS;
  - obtain seeds from another method (e.g. first do hierarchical clustering).

Number of clusters

- The number of clusters K is predefined.
- The ideal K will minimise variance within each cluster as well as minimise the number of clusters.
- There are various approaches to finding K. Examples that use techniques we have already learnt:
    - cross-validation;
    - PCA.

Number of clusters

- **Cross-validation:** split the data into random folds and cluster with some *k* on one fold. Repeat on the other folds. If points consistently get assigned to the same clusters (if membership is roughly the same across folds) then *k* is probably right.
- **PCA:** there is no systematic relation between principal components and clusters, but heuristically checking how many components account for most of the data's variance can give a fair idea of the ideal cluster number.

Evaluation of clustering

- Given a clustering task, we want to know whether the algorithm performs well on that task.
- E.g. clustering concepts in a distributional space: *cat* and *giraffe* under ANIMAL, *car* and *motorcycle* under VEHICLE (Almuhareb 2006)
- Evaluation in terms of ‘purity’: if all the concepts in one automatically-produced cluster are from the same category, purity is 100%.

Purity measure

- Given a cluster *k* and a number of classes *i*, purity is defined as

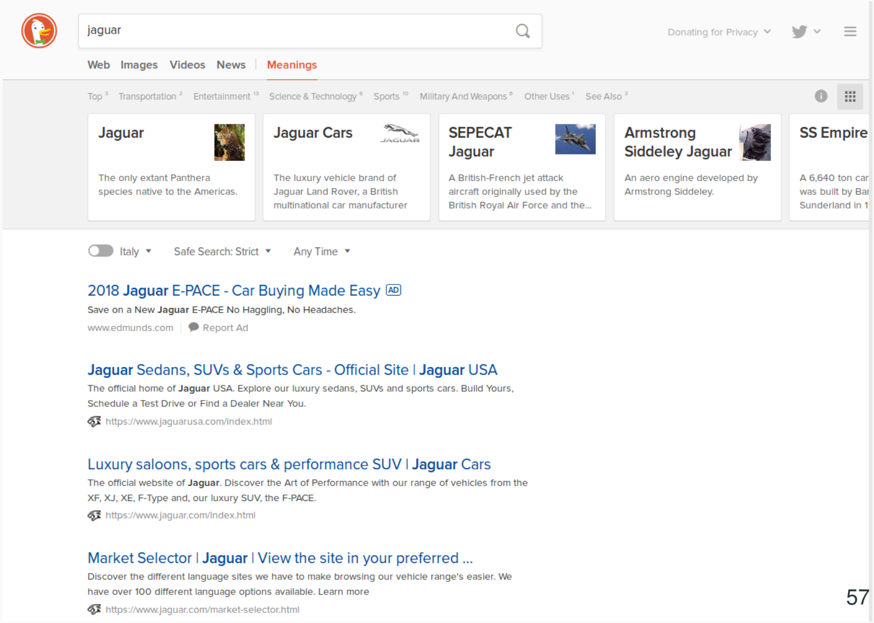
$$P(C_k) = \frac{1}{n_k} max_i(n_k^i)$$

- Example: given the cluster C_1 {A A A T A A A T T A}:

$$P(C_1) = \frac{1}{10} max(7, 3) = 0.7$$

- NB: here, the annotated data is only used for evaluation, *not for training!!* (Compare with k-NN algorithm.)

Clustering in real life



jaguar

Donating for Privacy

Web Images Videos News Meanings

Top 3 Transportation 2 Entertainment 13 Science & Technology 8 Sports 10 Military And Weapons 6 Other Uses 1 See Also 3

Jaguar
The only extant Panthera species native to the Americas.

Jaguar Cars
The luxury vehicle brand of Jaguar Land Rover, a British multinational car manufacturer

SEPECAT Jaguar
A British-French jet attack aircraft originally used by the British Royal Air Force and the...

Armstrong Siddeley Jaguar
An aero engine developed by Armstrong Siddeley.

SS Empire
A 6,640 ton car was built by Bar Sunderland in 1

Italy Safe Search: Strict Any Time

2018 **Jaguar** E-PACE - Car Buying Made Easy AD
Save on a New **Jaguar** E-PACE No Haggling, No Headaches.
www.edmunds.com Report Ad

Jaguar Sedans, SUVs & Sports Cars - Official Site | **Jaguar** USA
The official home of **Jaguar** USA. Explore our luxury sedans, SUVs and sports cars. Build Yours, Schedule a Test Drive or Find a Dealer Near You.
https://www.jaguarusa.com/index.html

Luxury saloons, sports cars & performance SUV | **Jaguar** Cars
The official website of **Jaguar**. Discover the Art of Performance with our range of vehicles from the XF, XJ, XE, F-Type and, our luxury SUV, the F-PACE.
https://www.jaguar.com/index.html

Market Selector | **Jaguar** | View the site in your preferred ...
Discover the different language sites we have to make browsing our vehicle range's easier. We have over 100 different language options available. Learn more
https://www.jaguar.com/market-selector.html